



Corso di Laurea in Ingegneria Civile

Corso di Elementi di Informatica

A.A. 2016-17

Docente: Alessandro Amirante

alessandro.amirante@unina.it

Università degli Studi di Napoli Federico II

Facoltà di Ingegneria



Agenda

- Costanti in C++
- Visibilità delle variabili
- Tempo di vita delle variabili
- Le stringhe in C++



Valori costanti

- In molti casi è utile assegnare a degli identificatori dei valori che restino costanti durante tutto il programma e che non possano essere cambiati nemmeno per errore
- In C++ è possibile ottenere ciò in due modi, con due risultati leggermente diversi:
 - Con il modificatore `const`
 - Con la direttiva al compilatore `#define`



Il modificatore `const`

- Una variabile a cui si antepone il qualificatore **`const`**:
 - Si inizializza in fase di dichiarazione
 - Non si può modificare nel corso del programma
 - Provoca allocazione di memoria

```
const float pigreco = 3.14159265;
```



Direttiva #define

- La direttiva:
 #define NOME valore
produce la sostituzione di tutte le successive occorrenze di **NOME** con **valore**
- **NOME** è un identificatore (tipicamente scritto in maiuscolo)
- **valore** è una stringa di caratteri
- Ad esempio, nella definizione delle dimensioni degli array:

```
#define MAX_LEN 1000
```

```
...
```

```
int a[MAX_LEN], b[2*MAX_LEN];
```

è equivalente a scrivere:

```
...
```

```
int a[1000], b[2*1000];
```



Direttiva #define

- Vantaggi:
 - Le costanti compaiono sotto forma di un nome simbolico
 - Il valore delle costanti può essere cambiato modificando solo la direttiva #define corrispondente
 - Il programma risulta *parametrico* rispetto alle costanti che usa



Direttiva #define

- È una delle direttiva al *precompilatore*
- Il valore viene sostituito testualmente prima che il programma venga compilato
- Non si può modificare nel corso del programma
- Non provoca allocazione di memoria
- Non viene terminata dal punto e virgole (non è una istruzione C++)



Direttiva `#define`

```
#define PIGRECO 3.141592
```

- **PIGRECO** non è una variabile
- La direttiva serve al precompilatore per informarlo che ogni successiva occorrenza del nome **PIGRECO** va sostituita con il numero **3.141592**
- In fase di compilazione questa direttiva viene filtrata, ovvero vengono ricercate tutte le occorrenze del **#define** e sostituire con il valore appropriato
- **const** e **#define** sono due cose diverse: di una si occupa il compilatore, dell'altra il precompilatore



Scope (visibilità) delle variabili

- La dichiarazione di una variabile o di un qualsiasi altro identificatore si estende dal punto immediatamente successivo la dichiarazione fino alla fine del blocco di istruzioni in cui è inserita
 - Un blocco di istruzioni è racchiuso sempre tra una coppia di parentesi graffe
- Ciò vuol dire che una dichiarazione non è visibile all'esterno del blocco in cui è dichiarata, mentre è visibile in eventuali blocchi annidati dentro quello dove la variabile è dichiarata



Scope (visibilità) delle variabili

```
{ ... // Qui X non è visibile
  int X = 5; // Qui X non è visibile
  // Da ora in poi esiste
  // una variabile X
  ...

  { // X è visibile anche in
    // questo blocco
    int a = 0;
    ...
  }
  ...
} // X ora non è più visibile
```



Scope (visibilità) delle variabili

- All'interno di uno stesso blocco non è possibile dichiarare più volte lo stesso identificatore, ma è possibile ridichiararlo in un blocco annidato
- In tal caso la nuova dichiarazione nasconde quella più esterna che ritorna visibile non appena si esce dal blocco ove l'identificatore viene ridichiarato
- All'uscita dal blocco più interno l'identificatore ridichiarato assume il valore che aveva prima di essere ridichiarato



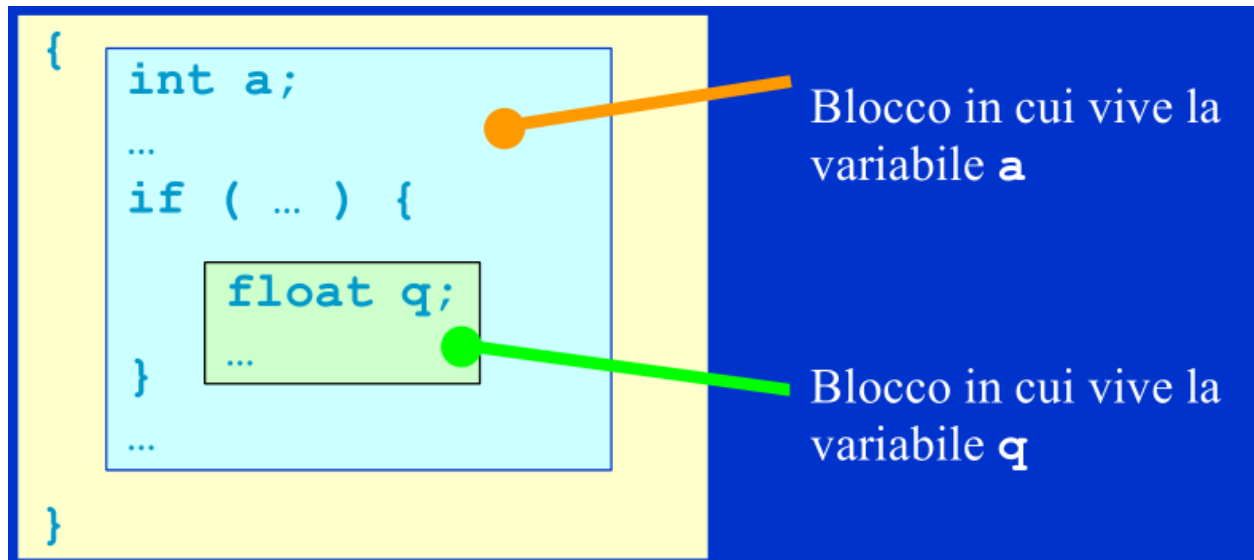
Scope (visibilità) delle variabili

```
{
    ...           // qui X non è visibile
    int X = 5;
    ...           // qui è visibile int X
    { ...         // qui è visibile int X
        char X = 'a'; // ora è visibile char X
        ...       // qui è visibile char X
    }             // qui è visibile int X
    ...
}                 // X ora non più visibile
```



Lifetime (tempo di vita) delle variabili

- Ogni variabile oltre a possedere uno *scope*, ha anche un propria durata (*lifetime*): viene creata subito dopo la dichiarazione (e prima dell'inizializzazione) e viene distrutta alla fine del blocco dove è posta la dichiarazione



- Le variabili *locali* (ovvero quelle dichiarate all'interno di un blocco) vengono create ogni volta che si giunge alla dichiarazione, e distrutte ogni volta che si esce dal blocco; è tuttavia possibile evitare che una variabile locale (detta anche *automatica*) venga distrutta all'uscita dal blocco facendo precedere la dichiarazione dalla keyword **static**



Variabili static

```
void func() {  
    int x = 5;           // x è creata e  
                        // distrutta ogni volta  
    static int c = 3;   // c si comporta  
                        // in modo diverso  
    /* ... */  
}
```

- La variabile `x` viene creata e inizializzata a 5 ogni volta che `func()` viene eseguita e viene distrutta alla fine dell'esecuzione della funzione
- La variabile `c` invece viene creata e inizializzata una sola volta, quando la funzione viene chiamata la prima volta, e distrutta solo alla fine del programma
- Le variabili statiche conservano sempre l'ultimo valore che viene assegnato ad esse e servono per realizzare funzioni il cui comportamento è legato a computazioni precedenti (all'interno della stessa esecuzione del programma) oppure per ragioni di efficienza
- La keyword `static` modifica il lifetime ma non modifica lo scope



Le stringhe di caratteri

- In generale, il termine stringa di caratteri indica una sequenza di caratteri atta a rappresentare una frase, un nome, un testo, un codice ed in generale una sequenza di simboli per il trattamento di testi
- Un valore costante di una stringa è racchiuso fra due delimitatori: **"questa e' una stringa di valore costante"**
- Il tipo stringa è di fondamentale importanza per tutte le elaborazioni non numeriche, come ad esempio quelle di I/O



Le stringhe in C++

- Gli array consentono la memorizzazione di stringhe
- In C++, una stringa viene definita come un array di caratteri che termina con il carattere NULL, specificato come `\0` e avente valore nullo
- A causa del valore NULL, è necessario che un array di caratteri venga dichiarato più grande di un'unità rispetto alla stringa più lunga che deve contenere
- Ad esempio, per dichiarare una stringa che dovrà contenere massimo 10 caratteri:

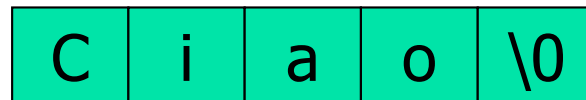
```
char str[11];
```

in questo modo si lascia lo spazio per il terminatore NULL alla fine della stringa



Le stringhe in C++

- Una costante stringa è una lista di caratteri racchiusa tra virgolette:
 - "ciao"
 - "12! \$£-+"
- La stringa "" prende il nome di stringa nulla
- Non è necessario aggiungere manualmente il terminatore NULL alla fine della stringa, in quanto tale operazione viene eseguita direttamente dal compilatore
- Per esempio, la stringa "Ciao" sarà registrata in memoria come segue:



- I comandi per la manipolazione di stringhe terminano quando trovano il carattere \0 (carattere *tappo*)



Lettura di stringhe in C++

- Il sistema di I/O del C++ termina la lettura di una stringa appena incontra un carattere di spaziatura (ritorno a capo, spazio, tab)
- Pertanto, non è possibile realizzare la lettura di una stringa che contenga degli spazi mediante l'utilizzo dell'oggetto `cin`
- È possibile utilizzare la funzione `cin.getline()`, cui è necessario passare l'array che dovrà contenere la stringa e la sua cardinalità
- N.B.: il comando `cin`, a differenza di `cin.getline()`, non svuota il buffer della tastiera al termine dell'operazione di input. Pertanto, per un uso combinato dei comandi `cin` e `cin.getline()`, è necessario far seguire al primo l'istruzione `cin.ignore()`



Lettura di stringhe in C++

```
int main() {
    const int DIM = 30;
    char frase[DIM];

    cout << "inserisci una frase: ";
    cin >> frase;
    cin.ignore();

    cout << "la frase inserita e': " << frase << endl;

    cin.getline(frase, DIM);

    cout << "la frase inserita e': " << frase << endl;

    system("PAUSE");
    return 0;
}
```



La libreria string.h

- Libreria di sistema che offre alcune funzioni utili per la manipolazione di stringhe
- **strcpy(s1, s2)**
 - copia il contenuto della stringa s2 nella stringa s1, senza alterare s2
- **strcat(s1, s2)**
 - Aggiunge (concatena) s2 a s1, senza alterare s2
- **strcmp(s1, s2)**
 - confronta le stringhe s1 e s2, restituendo 0 se sono uguali
- **strlen(s)**
 - restituisce la lunghezza della stringa s



Esempio 1

```
int main () {  
    char str[80];  
    strcpy (str, "ciao");  
    cout << str << endl;  
  
    system ("PAUSE");  
    return 0;  
}
```



Esempio 2

```
int main () {  
    char s1[20], s2[20];  
    strcpy (s1, "ciao");  
    strcpy (s2, " a tutti");  
    strcat (s1, s2);  
    cout << s1 << endl;  
  
    system ("PAUSE");  
    return 0;  
}
```



Esempio 3

```
int main () {  
    char str[80];  
    cout << "Inserire una stringa: ";  
    cin.getline(str, 80);  
  
    cout << "La lunghezza e': " << strlen(str);  
  
    system("PAUSE");  
    return 0;  
}
```

Domande?

